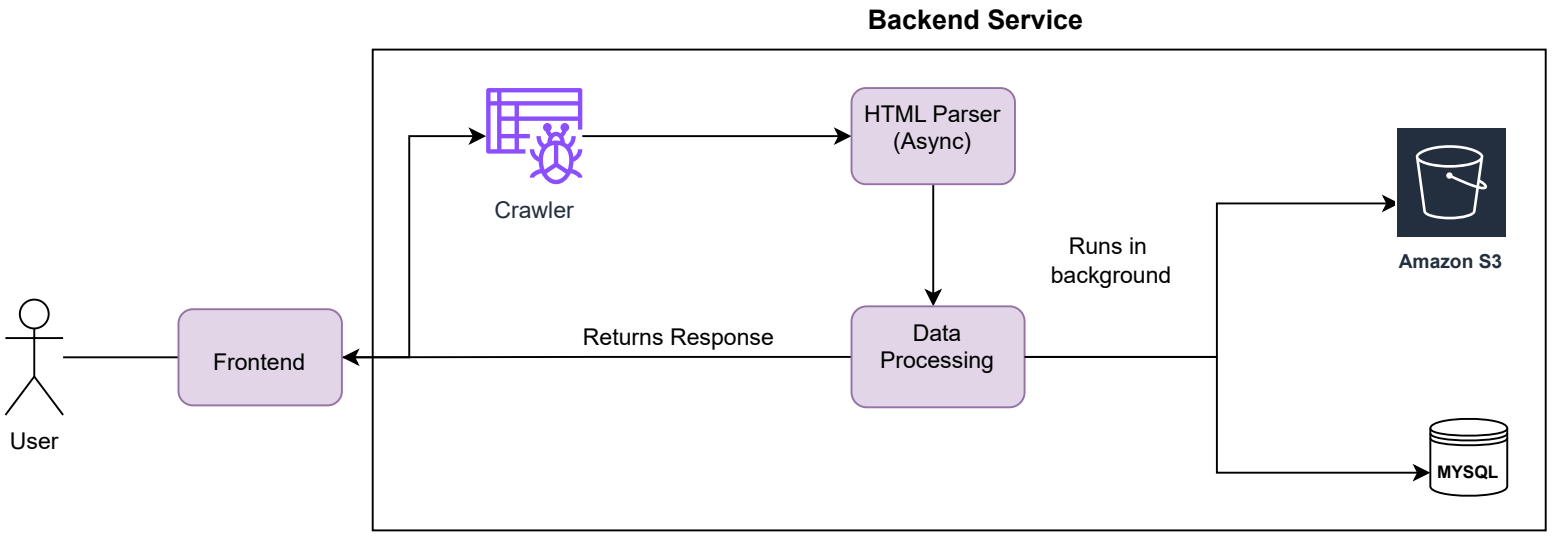

Data Extraction Using Crawler

1. User provides the URL(s) to the frontend service. Frontend is a separate service responsible for sending data extraction requests to the backend service. Frontend service can be built in python (using streamlit app).
2. The backend service has a crawler which scans through web pages, parses the data and processes the data to extract topics of interest (title, description, images) etc.
3. Post data processing, we respond the extracted data back to the client (frontend) and for storage push the extracted data to a message queue. The queue can be used to store the data into MYSQL (for storing text) and AWS S3 (for storing images) asynchronously.
4. The backend API sends a json response which is then converted to a structured format by the frontend service. The json response can look like:

```
{
'Url1': {
'title': 'product_title',
'description': 'some description',
'Images': ['base64_encoded_string_image_1', 'base64_encoded_string_image_2',
]}}
```

5. For deployment, I would set monitoring, logging and altering to track service performance and failures.

Tools and Services Used

1. Crawler: BeautifulSoup Python Library
2. HTML Parsing: Also, we can use beautifulSoup to do basic HTML parsing. If the parsing isn't correct, we can also use OpenAI APIs to do parsing.
3. Data Processing: Using regex/nltk/spacy for text cleaning. Pillow for image processing.
4. Backend API: It can be built in python using FastAPI.
5. Frontend API: Javscript/Streamlit Library Python
6. Deployment: Docker, CI/CD Pipeline, Kubernetes

Potential Challenges

1. Website Structure Variability:

Challenge: Websites may have varying HTML structures, CSS classes, and JavaScript frameworks, making it challenging to consistently extract product information.

Strategy: Implementing flexible parsing logic using robust HTML parsing libraries like BeautifulSoup or Scrapy, and continuously monitoring and updating parsing rules to adapt to changes in website structures could be helpful.

Scalability and Performance:

Challenge: As the system processes a large volume of web pages and images, scalability and performance become critical factors, especially during peak usage periods.

Strategy: Designing the system with scalability in mind, utilizing distributed processing frameworks, caching mechanisms, and load balancing techniques. Employing parallelization to optimize resource utilization and response times could help to resolve the issue.

Image Transfer:

Challenge: base64 encoded might increase the size of json response. Increased response size might take it longer for the client to display results on the frontend.

Strategy: Save the images at some external storage (CDN) and use the image URL in the response instead.